PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 15/177, 17/30

A1

(11) International Publication Number: **WO 00/45286**

(43) International Publication Date: 3 August 2000 (03.08.00)

(21) International Application Number: PCT/US00/02284

(22) International Filing Date: 28 January 2000 (28.01.00)

(30) Priority Data:

09/239,100	28 January 1999 (28.01.99)	US
09/339,724	24 June 1999 (24.06.99)	US

(81) Designated States: AU, CA, JP, KR, MX, SG, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(71) Applicant: GENRAD, INC. [US/US]; 7 Technology Park Drive, Westford, MA 01886–0033 (US).

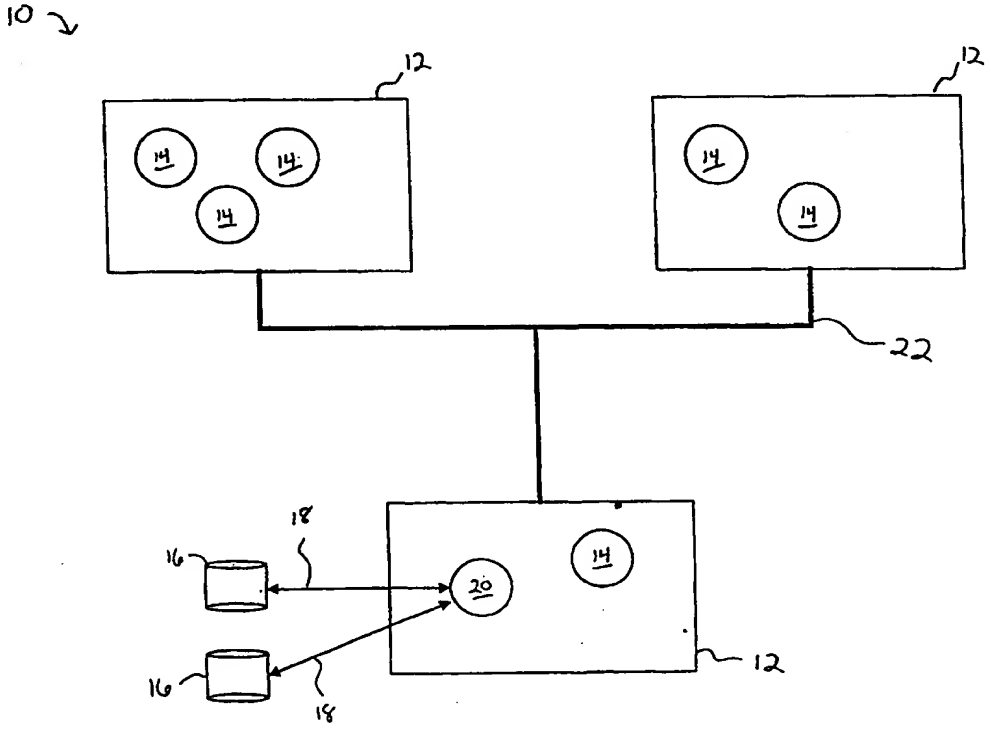
(72) Inventor: GIGLIOTTI, Samuel, Scott; 395 Birch Rill Drive, Alpharetta, GA 30022 (US).

(74) Agents: CAHILL, Ronald, E. et al.; Nutter, McClennen & Fish, LLP, One International Place, Boston, MA 02110–2699 (US).

(54) Title: METHOD AND APPARATUS FOR DISTRIBUTED DATABASE ACCESS

(57) Abstract

In a distributed computing environment having a plurality of computers connected by a communications network (10), a distributed database access system, method and computer program product including a plurality of clients represented by software running on a computer (12) connected to the communications network (22), a plurality of databases and a connection manager (16). The connection manager maintains a plurality of database connection pools with each pool maintaining one or more database connections to a database (18). Upon request from a client for database access (20), the connection manager places the client in communication with a database connection selected from a particular database connection pool (14). The plurality of databases can include at least one database storing data in a first data storage schema and at least one database storing data in a second data storage schema and the client request for database access can include a reference to a data storage schema. The invention may also be implemented within a transaction processing system.



METHOD AND APPARATUS FOR DISTRIBUTED DATABASE ACCESS

FIELD OF THE INVENTION

The invention provides a method and apparatus for providing database access in a distributed object computing environment. The database access system is particularly useful for providing database access in a transaction processing system.

BACKGROUND OF THE INVENTION

Consistency is a fundamental design goal of any transaction processing system. For example, when transferring money from a savings account to a checking account, the total of the two accounts must remain the same. If the savings account is debited but the checking account is not credited, the customer will be dissatisfied. On the other hand, if the checking account is credited but the savings account is not debited, the bank will become concerned.

A transaction is a collection of operations that performs a single logical action in a database application. In the above example, conducting an account transfer is a transaction. Debiting the savings account is one operation in the account transfer transaction, and crediting the checking account is another operation. In order to preserve consistency, every operation of a transaction must succeed or the entire transaction must fail -- that is, transactions can leave no work partially done. This requirement is called "atomicity."

Under normal conditions, consistency can be enforced in a transaction processing system by simply carrying out every operation in the transaction. However, software bugs, hardware crashes, and power outages can cause a computer system to fail. When a failure occurs, some information, particularly information that is stored in volatile memory such as RAM, may be lost and consistency violated. For example, a banking transaction processing system might debit the savings account but crash before crediting the checking account. To guarantee consistency, a transaction processing system must ensure that all of the operations of a transaction are executed or, if one or more of the operations that make up the transaction fails, none of the operations of the transaction are executed.

If a transaction cannot be successfully completed, and only some of the operations are executed, then the transaction must be "rolled back" and the completed operations undone. On the other hand, if all the operations in a transaction successfully execute,

then the transaction is "committed" and all of the operations are permanently stored in a database.

A distributed transaction is one in which operations occur on several different computers or in several different processes in a computer network. For example, checking account operations might occur on a first computer, savings account operations might occur on a second computer, and the request to transfer funds from a savings account to a checking account might originate at a third computer used by a bank teller. Every process or computer that is involved in the transaction is called a "participant." In a distributed transaction, partial failures can occur in which some computers are working while others are not, or where the computers are working but communication links between the computers have failed. Partial failures can also occur where one operation fails because it violates a logical restraint, either in the system or on its local computer.

The benefits of a distributed transaction system include improved performance and scalability, that is, the ability to support additional users by adding additional computers without losing performance. Because failures in a distributed system are usually partial ones, another benefit is improved reliability — if one computer fails, often the system continues to operate. However, the possibility of a partial failure makes the assurance of consistency more problematic. For example, the computer that processes savings account operations might function normally, deducting the debit, whereas the computer that processes the checking account operations might fail without adding the credit.

In order to preserve consistency, transaction processing systems implement a "commitment protocol." One common commitment protocol for distributed transactions is the "two-phase commit" protocol. A two-phase commit protocol generally includes two phases: a prepare transaction phase and a resolve transaction phase.

First, in the prepare transaction phase, the creator of a transaction asks each participant to prepare or abort the transaction. Each participant must determine whether it wishes to commit or abort the transaction. If the participant wishes to commit the transaction, it records the fact that the transaction is prepared for commitment to its local transaction log in non-volatile storage. The local transaction log will have already recorded the old and new values of the local changes made by that transaction to the database. Then the participant sends a "commit" vote back to the owner.

If the participant decides to abort, it records an abort of the transaction to non-volatile storage and sends a "roll back" vote back to the owner. There are a number of reasons why a participant might decide to abort. An operation may violate some constraint imposed on the logic of that operation. For example, if debiting the savings account would reduce the balance in the savings account below zero, then that participant would abort the transfer transaction.

Second, in the resolve transaction phase, the creator collects all the votes from the participants. If all the participants voted yes, then the owner records a commit of the transaction to its transaction log in non-volatile storage. At this point the transaction is committed. Then the owner sends a message to each participant to commit the transaction.

If any participant voted to roll back the transaction, then the creator records an abort of the transaction to non-volatile storage, and sends a message to each participant to roll back the transaction. Each participant that placed a prepared to commit record in non-volatile storage will wait for a commit or roll back message from the creator to take action.

One important application for a distributed transaction processing system is the Manufacturing Execution System ("MES"). Generally, an MES is an enterprise-wide software application integrated with one or more shop floors in a manufacturing company. Goals of an MES include sending timely shop floor data to the enterprise and incorporating enterprise-wide requirements into the shop floor.

Ideally, an MES should collect data from a variety of factory floor devices such as SCADA, controllers, test equipment, bar code scanners and others. An MES should also allow definition of bills of material and specification of routing steps and operation sequences. An MES can also define shop floor equipment, including its capabilities and requirements, and manufacturing employees, including their training and certifications. Output from an MES can include order tracking, work-in-process traceability, defect and rework management, work instruction management, data collection and statistical process control and statistical quality control ("SPC/SQC"), process licensing, item qualification, labor tracking and finite scheduling. An MES thus improves management's visibility into its manufacturing processes and interacts with enterprise resource planning and supply chain management software and professionals to reduce manufacturing lead times and meet customer and manufacturing deadlines.

Generally, transaction processing systems store data to databases in an "on-line transaction processing" (OLTP) format. Transaction processing supports business functions such as order entry, accounts payable and the MES functions described above. The most common activity in such a system is the recording of daily data transactions. Transaction data tends to be very detailed, corresponding to the lower levels of data collection in an enterprise. For example, transaction data generated by a "defect" transaction in an MES system where an automated tester finds a defect in a circuit board may include the time and date that the defect is found, the part number of the circuit board, and a code representing the defect detected.

OLTP systems are generally designed to ensure that current data is stored in an efficient and reliable manner. The data in OLTP systems is generally flat or file-oriented; it consists of linear lists of related values, each describing an employee, customer, product, or other entity. Most of the time, this data is accessed using a single key value and is viewed as a simple list on the screen or page. OLTP data is typically highly normalized.

Despite the advantages of OLTP systems, many businesses find such systems wanting. Business managers need to have more than a data store, they need to be able to query the data in a manner that is relevant to the ongoing issues in the enterprise. Referring to the "defect" transaction example above, a manager is not content simply to know that a defect occurred, but must know of trends in the defect data that have an impact on the business. Do certain vendors supply circuit cards with higher defect rates? Do certain shifts in the manufacturing operation produce more defects than others? Are defects more likely to occur at a particular time of day or on a particular day of the week? All of these queries are important to the business's goal of reducing or eliminating defects, but OLTP systems are not well suited to these types of queries. These types of queries are best handled by an "on-line analytical processing" (OLAP) system. While OLTP systems provide highly efficient execution of a large number of small transactions with near fault tolerant availability of data, OLAP systems transform raw data so that it reflects the real dimensionality of the enterprise as it is understood by the user who queries the system.

Rather than the traditional flat-file approach of OLTP systems, OLAP systems take a multidimensional approach to data. Often, OLAP databases are structured as a "data cube" using a star schema or a database structure closely resembling a star schema.

By structuring the database to provide dimensional views that represent the underlying business, OLAP systems can provide highly efficient execution of complex dimensional queries with a minimum of database query programming. Unfortunately, the OLTP systems that already contain much of an enterprise's data and which are necessary to track transactions in the enterprise, do not store data in this manner.

One attempt to bridge the gap between OLTP and OLAP systems has been to develop OLAP-type query tools for OLTP databases. In general, these query tools are inefficient and difficult to program due to structure of the underlying database. As a result, the performance of these tools has been limited and disappointing.

Another attempt to bridge the gap between OLTP and OLAP systems has been to develop tools that convert, typically at regular time intervals, data from OLTP databases into an OLAP format in a separate database. OLAP query tools can then be used to efficiently mine the transformed data. These systems lack timeliness, however, as the OLAP data always lags behind the OLTP data. In addition, the quality of the OLAP query ability depends upon the quality of the data conversion from flat-file to multidimensional format and these conversions are as difficult to program as OLAP queries for flat files. The actual conversion process is also very computer resource and time demanding.

In order to solve these problems and support businesses that need OLTP systems to track low level data as well as OLAP-type query ability, it would be desirable to provide a database access system that can produce data in both flat-file and multidimensional formats in a timely and efficient manner. It would also be desirable to provide the database access system having this capability as part of a transaction processing system. In such a system, transaction processing safeguards can be employed to ensure consistency in each type of data store as well as between the data stores.

SUMMARY OF THE INVENTION

To address the shortcomings in known systems, the invention provides a method, system and computer program product for distributed database access in a distributed computing environment.

According to the invention, the distributed database access system is provided in a distributed computing environment having a plurality of computers connected by a communications network. The system includes a plurality of clients represented by

software running on a computer connected to the communications network and a plurality of databases. The distributed database access system further includes a connection manager maintaining a plurality of database connection pools where each pool includes one or more connections to one of the plurality of databases. When a client requests database access, the connection manager places the requesting client in communication with a database connection selected from a particular database connection pool.

In one embodiment, each database connection pool maintains one or more database connections to a single database and the plurality of databases includes at least one database storing data in a first data storage schema and at least one database storing data in a second data storage schema. Clients can then include a reference to a particular data storage schema with their request for database access and the connection manager can convert that request into a reference to a database connection to a database that stores data in the requested schema. The different data storage schemas employed by the databases can include a schema for on-line transaction processing and a schema for on-line analytical processing.

The database access system of the invention can be employed within a transaction processing system. Within such a system, at least one of the plurality of clients participates in a transaction by registering with a transaction context. One or more clients participating in the transaction can request database access and include in the request a reference to a particular data storage schema. Where transaction participating clients request database connections to databases storing data in different data storage schemas, the transaction processing system can ensure consistency between the different databases by employing a means for commitment of transaction data such as a two-phase commitment protocol.

The invention also provides a method for distributed database access in a distributed object computing environment having a plurality of computers connected by a communications network including a plurality of client objects running on the computers. The system also has a plurality of databases including at least one database storing data in a first data storage schema and at least one database storing data in a second data storage schema. A connection manager in the system maintains a plurality of database connection pools with each pool maintaining one or more database connections to a database. In the method, a client object request databases access, including in the request a reference to the first or second data storage schema. The connection manager then

passes to the client object a connection to a database that stores data in the requested data storage schema.

The method can also be implemented in a distributed object transaction processing system wherein a transaction originating object initiates a transaction by initiating a transaction context. The transaction initiating object also publishes a message into the distributed object transaction processing system. One or more objects in the distributed object transaction receive the published message and register with the transaction context as transaction participants and one or more transaction participants place one or more requests for database access, including a reference to a data storage schema, with the connection manager. The connection manager responds to the requests by providing a reference to a database connection that fulfills the request to the transaction participant placing the request. Transaction participants having database connections can then place data into the connections as part of the transaction in which they participate. Upon closing of the transaction, the data in the database connections can be committed to the databases.

The invention also includes a computer program product comprising a computer useable medium having computer readable program code that implements the system and methods of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

The features of the invention may be more fully understood from the following detailed description of the drawings in which:

FIG. 1 illustrates a database access system of the invention having a database connection manager;

FIG. 2 illustrates the database connection manager of FIG. 1;

FIG. 3 illustrates a transaction processing system of the invention having a database connection manager;

FIG. 4 illustrates an additional embodiment of the transaction processing system of the invention;

FIG. 4A illustrates a Transaction Context and database connections useful with the transaction processing system of FIG. 4; and

FIG. 5 illustrates communications among objects using the transaction processing system of FIG. 4.

DETAILED DESCRIPTION OF THE INVENTION

The present invention provides a database access system for use in a distributed computing environment, and in particular, a distributed object computing environment. A distributed object computing environment 10 of the invention is illustrated in FIG. 1. The distributed object computing environment 10 includes a plurality of machines 12 with software objects 14 being executed on machines 12. One or more databases 16 are also connected to the distributed object computing environment 10. Database 16 may have a plurality of direct connections 18 to one or more objects 14 to interface to the distributed object computing environment 10. More particularly, a database connection manager object 20 may maintain one or more database connections 18 to one or more databases 16.

Object-oriented computer programming techniques involve the definition, creation, use and destruction of software entities referred to as "objects." Each object is an independent software entity comprised of data generally referred to as "attributes" and software routines generally referred to as "member functions" or "methods" or "logic" which manipulate the data.

One characteristic of an object is that only methods of that object can change the data contained in the object. The term "encapsulation" describes the concept of packaging the data and methods together in an object. Objects are thus said to encapsulate or hide the data and methods included as part of the object. Encapsulation protects an object's data from arbitrary and unintended use by other objects and therefore prevents an object's data from corruption. Encapsulation also 'hides' the implementation details, thereby supporting a wide range of possible implementations that result in the same data services and behaviors.

To write an object-oriented computer program, a computer programmer writes computer code that implements a pre-defined model of the system which typically represents 'real world' elements and their relationships. The object oriented-computer code defines a set of "object classes" or more simply "classes." Each of these classes serves as a template which defines a data structure for holding the attributes and program instructions which perform the method of an object. Each class also includes a means for instantiating or creating an object from the class template. The means for creating is a method referred to as a "constructor." Similarly, each class also includes a means for

destroying an object once it has been instantiated. The means for destroying is a method referred to as a "destructor."

When a processor of a computer executes an object-oriented computer program, the processor generates objects from the class information using the constructor methods. During program execution, one object is constructed, which object may then construct other objects which may, in turn, construct other objects. Thus, a collection of objects which are constructed from one or more classes form the executing computer program.

Object-oriented computer programming techniques allow computer programs to be constructed of objects that have a specified behavior. Several different objects can be combined in a particular manner to construct a computer program which performs a particular function or provides a particular result. Each of the objects can be built out of other objects that, in turn, can be built out of other objects. This resembles complex machinery being built out of assemblies, subassemblies and so on. Similarly, computer programs can be assembled from different types of objects each having specific structural and functional characteristics.

The term "client object," or more simply "client," refers to any object that uses the resources of another object which is typically referred to as the "server object" or "server." The term "framework" can refer to a collection of inter-related classes that can provide a set of services (e.g., services for network communication) for a particular type of application program. Alternatively, a framework can refer to a set of interrelated classes that provide a set of services for a wide variety of application programs (e.g., foundation class libraries for providing a graphical user interface for a Windows system). A framework thus provides a plurality of individual classes and mechanisms which clients can use or adapt. In one embodiment, the system of the invention is supplied as a framework that a business enterprise can tailor to its particular needs.

An application framework refers to a set of classes which are typically compiled, linked and loaded with one particular application program and which are used by the particular application program to implement certain functions in the particular application program. A system framework, on the other hand, is provided as part of a computer operating system program. Thus, a system framework is not compiled, linked and loaded with one particular application program. Rather, a system framework provides a set of classes which are available to every application program being executed by the computer system which interacts with the computer operating system.

Generally, a distributed computer network 10 for implementing the system and method of the invention may include any number of computer "machines" 12 which may be actual computers, such as work stations or PCs commonly known in the art or any other computers useful as either server or client machines, or virtual machines. Virtual machines are software devices that operate on a computer but appear to the software that runs on the virtual machine to be a complete computer. A common example of a virtual machine known in the art is the Java virtual machine, however, other types of virtual machines are available and may be used herein. Computers in a computer network implementing the systems and methods of the invention can be connected by a communications network 22 such as a TCP/IP network (including intranets or the Internet), SQLNet, or the like.

Software objects 14 generally run on machines 12 in a distributed computer network 10. Objects 14 may be created using any object-oriented software programming language known in the art such as C++, Java, Objective C, SmallTalk or others. Additionally, a number of visual and textual software development tools exist to help developers to define and create objects, including Visual C++, Rational Rose, and Persistence. One or more databases may also be provided on one or more machines. The databases 16 may be object-oriented databases or relational databases. The description that follows will generally refer to databases used with the invention as relational databases, however, a person of ordinary skill in the art will recognize that object-oriented databases may be used in place of the relational databases.

The systems and methods of the invention may also be implemented in a distributed object computing environment 10 that uses a communications network 22 known as an Object Request Broker ("ORB"). An ORB is middleware that manages communications and data exchanges between objects, even when those objects are on different machines. The primary functions of an ORB include defining interfaces between objects, locating and activating remote objects, and communications between clients and objects. The goal of an ORB is to make it appear as if an object is local to a client, while in fact the object may reside in a different process or machine. A variety of ORB standards are available for software development including COM/DCOM by Microsoft, the Common Object Request Broker Architecture ("CORBA") organized by the Object Management Group ("OMG"), and the Remote Method Invocation ("RMI") extensions to the Java language/virtual machine. Distributed object applications can be

particularly useful for deploying object-oriented software throughout a geographically disparate enterprise while maintaining the performance and feel of a single application to users at any location.

Communications network 22 may also include a publish-subscribe event protocol. In publish-subscribe communications, objects may publish information or requests of interest and other objects may subscribe to a selected subset of published material or events. Publish-subscribe or event-driven communications provide asynchronous communication among objects, that is, a publishing object need not wait for, or even know about, subscribing objects in order to publish. As an example of one possible publish-subscribe protocol, OMG has developed a standardized specification for event-based publish-subscribe communications and CORBA now includes a "CORBA Event Service" specification that includes a "CORBA Event Channel."

Consistent with the distributed object model, connection manager 20 may be located on any machine connected to the communications network 22 and need not be located in proximity to a database 16 to which a connection is made. Connection manager 20 may be a single class that manages a pool 24 of database connections 18. The database connections 18 themselves may be instances of a database connection class, may be addressable across an ORB and may allow connections to a number of different types of databases 16. Database connections 18 may be created, for example, using Java Database Connectivity ("JDBC"). JDBC provides classes that enable Java clients to interact with relational databases compliant with Microsoft's Open Database Connectivity ("ODBC"). It includes facilities to open and close database connections, query metadata information, issue SQL statements, and get result sets and other database-related operations. Accordingly, connection manager 20 may manage relational database connectivity from anywhere within the distributed computing environment. In addition, more than one connection manager 20 may be employed in order to provide increased performance.

Connection manager 20 is illustrated in further detail in the embodiment shown in FIG. 2. Connection manager 20 includes a number of database connection pools 24a-f. Six pools 24 are shown, however, more or fewer connection pools may be employed as needed. Each database connection pool 24 maintains a number of database connections 18.

The number of connections 18 in a pool 24 is not particularly limited, however, in a preferred embodiment, the number of database connections 18 provided by a database connection pool 24 may grow or shrink according to the level of usage of the database connection pool. For example, if a database connection pool 24 receives a request for a database connection 18 at a time when all of the existing database connections in that database connection pool are in use, the database connection pool may call a database connection constructor to create a new database connection to the desired database to handle the need for database access. Similarly, the database connection pool 24 can monitor the database connections 18 in its pool and can terminate connections that remain idle for a predetermined amount of time. In this manner, the database connection pool 24 can respond to the needs of client objects, yet conserve system resources by terminating unused connections.

In one embodiment, each database connection pool 24 provides connections to a single database 16. For example, database connection pool 24a provides database connections 18 to database 16a. In this manner, databases 16 may have different properties, they might be databases supplied by different vendors or store data in different schemas, and client objects may push data to particular databases 16 by connecting to the database through a particular database connection pool 24 that contains connections to the desired database. For example, an administrator object within the distributed object computing environment 10 may maintain a table, such as Table 1 below, having a list of client classes and defining primary, archive, primary-backup and archive-backup databases. The numbers in Table 1 represent database connection pool nos. 1, 2, etc., each having connections 18 to a different database 16.

TABLE 1

Client Class	Primary	Archive	Primary-Backup	Archive-Backup
employee	1	3	4	6
defect	1	2	4	5
machine	1		4	

In the example of Table 1, objects of the class "employee" store "primary" data to the database associated with database connection pool number 1. Objects of the class "employee" store "archive" data to the database associated with database connection pool

number 3. As can further be seen from Table 1, the "defect" class uses the same primary, but a different archive database than the class "employee." The class "machine" in this example only stores "primary" data. For each class, "backup" databases can be designated in case, for example, a designated database is unavailable, the connection manager can automatically traverse to a backup database. Zero or more databases may be designated as backup databases.

If, as part of its logical operation, an object of the class "employee" wishes to store "primary" data, the object can request a database connection from the connection manager 20 by invoking a "get database connection" method from the connection manager and passing its class name and the fact that it wishes a connection for storing "primary" data. Based on the data in Table 1, connection manager 20 will pass to the client employee object a database connection 18 from database connection pool 1 24a to database 16a. In general, databases 16 designated as "primary" are flat-file relational databases designed for OLTP data storage. Databases of this type are readily available from commercial database vendors such as Oracle Corp. of Redwood Shores, California, Sybase, Inc. of Emeryville, California, and Microsoft Corp. of Redmond, Washington. Connection manager 20 may "pass" a database connection by responding to a "get database connection" method call by returning the CORBA Interoperable Object Reference ("IOR") for a database connection 18 from the appropriate database connection pool 24. An IOR is specifically intended to convey object reference protocol information across ORB and inter-ORB boundaries. Using the IOR, the client object can interact directly with a specific database connection 18.

In another example, if a "defect" client object, as part of the operation of its internal logic, sought to push primary data to a database, it would request a primary database connection from connection manager 20 and would receive an IOR to a database connection 18 from Pool 1 24a to database 16a. The defect object could then submit basic defect transaction data such as the time, date, defect code and part number to the database 16a for OLTP storage.

The defect object could also, as part of its internal logic, seek to store defect information in a multidimensional OLAP format. The defect object could request an additional database connection 18 from the connection manager 20 as an "archive" connection. Connection manager 20 can then pass an "archive" connection to the defect object. Referring again to Table 1, the defect object stores archive data in the database

16b connected to database connection pool 2 24b. In order to store archive data, it is possible that the defect object would need to acquire additional information beyond that required for OLTP storage such as the vendor of the defective part, the shift during which the part was assembled, etc. In such an event, the defect object may obtain the required information by, for example, employing internal logic to acquire such information, perhaps by making appropriate database queries, or by invoking methods in other business objects that have or have access to such information.

A client object, such as the defect object described above, may store only primary data, only archive data, or by acquiring two database connections, may store both primary and archive data. Alternatively, a client object could store one type of data and invoke another object to store the other type of data. For example, a defect object could store primary data and invoke a method in a "defect archive" object which could then acquire additional information and execute the archive store. In this way, the OLTP data transaction can be performed efficiently and asynchronously from the OLAP data transaction. In addition, through the connection manager of the invention, objects may store data in functionally distinct databases. For example, an employee object may store primary data in the primary transaction data store, but could archive data to, for example, a human resources database.

OLAP and OLTP databases are typically physically separate databases and may even be supplied by different vendors. A variety of OLAP databases are commercially available, including, for example, the Oracle Express product line from Oracle Corp. of Redwood Shores, California. The connection manager 20 can be configured to hide these details from client objects. In this way, client objects need only know whether they are storing primary or archive data and the correct database connection will be passed to the client object for storing the data, the client object then implements the appropriate data format and sends it to the database using the database connection that has been passed to it. This allows a high degree of encapsulation of the business logic within the client objects and the details of the database connections within the connection manager 20.

Connection manager 20 may also include database connection pools 24 having database connections 18 to backup databases. The objects listed in Table 1 each list database pools for backup databases. If, for example, an employee object requests a primary database connection, connection manager 20 can test a connection 18 from pool 1 24a, the pool specified for employee primary storage. If the connection 18 is in

working condition, connection manager 20 can pass that connection to the employee object. If however, the connection 18 from the designated database connection pool 24 is inactive, due possibly to a failure in the connected database 18, connection manager 20 can automatically traverse to a backup database, using the reference to database connection pool 4 24d. In this way, the client object's data can be appropriately stored to a database 16 without the client object knowing whether the preferred database 16 is in working order. Database vendors typically supply backup database tools that allow for synchronization of backup data when the preferred database is restored to proper working order. Any number of backup databases may be specified for a client object and backup databases can be traversed in order until a working database is found.

The database access system of the invention may be implemented as part of a transaction processing system. A transaction processing system 100 of the invention, illustrated in FIG. 3, includes a plurality of machines, and while the number or location or type of machines are not particularly limited, for convenience, the machines in FIG. 3 may be referred to as a client machine 102, a server machine 104, and a database machine 106. Each of these machines is connected by a communications network 108 which may be an ORB.

Transaction processing system 100 also includes at least one client object 110 and a plurality of server objects 112. It will be understood, consistent with the use of ORB technology, that these objects may be located anywhere in the transaction processing system 100 and need not be affiliated with a specific machine. Transaction processing system 100 further includes at least one database 114. A connection manager 116 is provided to interface databases 114 with objects in the system 100 that may need database access or services. While FIG. 3 illustrates a transaction processing system 100 having a simplified connection manager 116, connection manager 116 preferably includes the structure and features of connection manager 20 (FIG. 2).

A transaction context object 120 is also used during the processing of a transaction. Transaction context 120 includes methods to add a participant in a transaction, register commit or rollback votes from participants, and can commit or rollback an entire set of operations and allow related objects access to uncommitted data. While the transaction context 120 may exist anywhere in the system 100, it may be convenient to provide the transaction context 120 on the same machine as the connection manager 116 given that a transaction that uses a transaction context 120 will also likely

need database 114 access in order to be completed. Each transaction will have one transaction context 120.

Client object 110 may be an ORB compliant object (such as a CORBA object) and have an associated Graphical User Interface ("GUI") to allow a system user to interact with the software to cause the client object to initiate a transaction. Alternatively, client object 110 may represent a client, for example a dedicated device, such as a bar code scanner, having an embedded processor or being connected to a general purpose computer that may be programmed with an object wrapper. For example, all CORBA objects are defined using Interface Definition Language (IDL). IDL is an object-oriented interface definition formalism that completely defines the interface between objects. Using IDL, a client may be made to appear to the system 100 as a CORBA object, regardless of the client's location or implementation.

A client object may initiate a transaction on action of a user interacting with GUI. For example, a system user may wish to add or delete an employee from the list of employees at a particular manufacturing site. The user would interact with a GUI to provide the required information, perhaps including the employee's training and certifications or the specific manufacturing team in which the employee will be included, and indicate that the user interaction is complete, likely by activating a virtual "save" or "complete" button with a mouse. Transactions may also be initiated, for example, by a dedicated device such as a bar code scanner. If, for example, an inspection process results in a defect indication for a particular bar coded work item, either by automatic operation of the manufacturing inspection equipment or by user interaction, the bar code on the work item may be scanned to indicate that this particular item is defective. Transactions may also be initiated by some combination of the above, for example a user might scan the bar code of a defective work item, input information about the defective item through a GUI, and indicate that the interaction is completed to initiate a transaction. In each of these examples, data has been entered into the transaction processing system 100 and the system must process this data to update the appropriate databases and inform whatever other portions of the system 100 that have a need to know the newly entered data. It is likely that the client, or user interacting with the client, does not know how the data will be used within the system 100.

To initiate a transaction, a client object 110 creates a transaction context 120, for example by calling a transaction object to create a process for the transaction, and

publishes an event. Creation of a transaction may be performed by a stand-alone transaction object, or by a method that may be called on the connection manager 116 to create a new transaction context 120. The event published by the client object 110 will include a reference to the transaction context 120, a reference to the client object 110, and data. Data, for example, may include the bar code scan for a defective work item along with any other information entered by the client about that work item. The client object 110 also registers with the transaction context 120 as a participant in the transaction. Client object 110 may also have data to add to the databases as part of the transaction. If the transaction is ultimately committed, this data will be stored to one or more database 114.

Zero or more server objects 112 in the transaction processing system 100 can be responsive to the event published by the client object 110. Typically, one or more server objects 112 are responsive to an event, however, for some events and under some circumstances, it may be that there are no responsive server objects and the transaction may include only the publishing object as a participant. A responsive server object may include, for example, an object that updates work-in-process data to account for work items removed from their ordinary work flow because they are defective. Such an object would "listen" for or subscribe to events relating to defective work items and could perform some logic, which might include publishing events of its own. The responsive server object also uses the transaction context 120 reference provided in the event to register with the transaction context as a participant in the event. As with client object 110, server object 112 may have data to add to the databases as part of the transaction. As part of its operation, server object 112 may request database services in order to supply data for application in the transaction.

After registering with the transaction context 120 and performing any required logic, the server object 112 places a vote with the transaction context 120. If the logical operations performed by the server object 112 are successfully completed, the server object will typically vote to commit the transaction. If the logical operations are not successfully completed, the server object 112 will typically vote to roll back the transaction.

Using the reference to the client object 110 included in the event, the server object 112 will also call back the client object to inform the client object that the server object has acted on the event published by the client object and to send to the client object any

data that forms an appropriate response to the event. For example, an event published by a client object 110 may include a request for data. A responsive server object 112 would perform the necessary logic to acquire the data and respond with that data to the client object 110 through a call back.

Once the client object 110 has received all expected call backs, the client object places its own vote with the transaction context 120 and requests that the transaction context close the transaction. The transaction context 120 will then review the votes placed for the transaction. If any transaction participant has voted to roll back the transaction, the transaction context 120 rolls back the transaction.

Where the client object 110 publishes a defect event, that object, referring to Table 1 and FIGS. 1 and 2 above, may store defect transaction data in a primary database 16. Alternatively, the client 110 may simply publish an event having the defect data and a server object 112 responsive to that event may store the data to a primary database 16. In addition, either the client 110, server 112, or another server object 112 may store the defect data in an archive database 16 along with any other data that the object may need to perform the archive storage. By implementing connection manager 20 (FIG. 2) within a transaction processing system such as system 100, different databases, such as primary and archive databases, can be updated in an asynchronous fashion by different objects while ensuring through the transaction processing commitment process that both databases will be updated or the transaction will be rolled back. In this way, consistency is maintained between the different databases.

FIG. 4 illustrates an exemplary embodiment of a transaction processing system 200 of the invention. Transaction processing system 200 includes a Transaction Context Participant Agent ("TCPA") 202. TCPA 202 is a class that provides the functionality necessary for an object to participate in a transaction. TCPA 202 may also provide the functionality required for a non-IDL object to participate in a transaction across an ORB. For example, where client 204 is not a CORBA object, TCPA 202 may provide the functionality required for the client 204 to interact across a CORBA ORB. The methods contained in TCPA 202 for participating in a transaction may include creating a new transaction context 206 and returning the IOR for the transaction context created. Transaction context 206 may be located on the same machine as connection manager 222, which maintains a pool of database connections 224 to one or more databases 226.

Again, connection manager 222 preferably includes the structure and features of connection manager 20 (FIGS. 1 and 2).

The object that creates a transaction context can be identified as the creator of a transaction by setting a "creator" flag in the TCPA. The creator of a transaction may have special privileges such as the ability to close the transaction. TCPA 202 may also include methods for adding the object having the TCPA to a transaction using the transaction context 206 IOR, adding data to the transaction, requesting database services, registering a commit or rollback vote with the transaction context, closing the transaction, checking to see if all participants have voted, getting a list of participants in the transaction and retrieving data from the transaction.

A more detailed view of Transaction Context 206 is provided in FIG. 4A. When a transaction participant such as client 204 or server object 220 registers with Transaction Context 206, if the participant requires a database connection 224 to store data as part of the transaction, the participant requests a database connection 224 from Transaction Context 206 when it registers. Transaction Context 206 requests the database connection 224 from Connection Manager 222 which provides the connection from a pool of connections that it maintains, or, if necessary, Connection Manager 222 creates a new database connection. Where the Client 204 requests a particular type of database storage, such as "primary" or "archive" (See Table 1), Transaction Context 206 passes that information along with the class name for the client object to Connection Manager 222 so that Connection Manager 222 returns a database connection to the desired type of database.

Once passed to Client 204 or server object 220, database connections 224 can be used to store uncommitted data as part of the transaction. Transaction Context 206 also retains a reference to each database connection 224 that is passed and the name of the participant that the connection was passed to in a Connection Table 223. Connection Table 223 may also have an entry to keep track of the type of database connection that is passed, such as "primary" or "archive." Transaction Context 206 also maintains a Vote Table 225 to track the vote placed by each participant and the participant name. Generally, participants will place a "no-vote" vote upon registration, and vote to commit or roll back the transaction after performing logical operations. When the transaction is closed, if each participant has voted to commit, Transaction Context 206 sends a commit data message to each database connection 224 listed in Connection Table 223 to commit

the uncommitted data and the database connections can be returned to the Connection Manager's 222 connection pool.

When client object 204 publishes an event, the event carries with it TCPA data such as the transaction context 206 IOR. In this manner, any object in the transaction processing system 200 that subscribes to the published event may register directly with the transaction context 206 and participate in the transaction.

Events may be published by a client 204 using event services provided in the transaction processing system 200 as illustrated in FIG. 4. Publisher of Events Agent ("PEA") 208, Event Publisher 210, Event Channel 212, Event Subscriber 214 and Subscriber for Events Agent ("SEA") 216 are all objects connected to a communications network. That is, these objects may be CORBA objects, but the objects themselves are part of the transaction processing system 200 and are not part of the ORB. As such, these objects can implement a multicasting and distributed callback system of messaging within the transaction processing system 200 above and beyond any messaging services provided as part of the ORB.

In an exemplary embodiment, a PEA 208 class provides a CORBA object that is called by a publishing client to publish an event. PEA 208 attaches a reference to itself to the event, such as an IOR that refers to itself, and sends the event to Event Publisher 210. PEA 208 also returns responses from objects that subscribe to the event to the publishing client.

Event Publisher 210 provides a way for objects to publish events in the distributed transaction processing system 200. Event Publisher 210 receives events from the publishing object and places them on the appropriate Event Channel 212. Events may conveniently be published as serialized objects. Event Publisher 210 may maintain a list of subscriber classes for specific event types. Lists of this type may be stored in an administrator object and requested by Event Publisher 210 as needed. Having this information, Event Publisher 210 can provide PEA 208 with a list of all subscribing classes upon publication of an event. Having this list, PEA 208 can determine whether it has received responses from each subscribing class.

In the transaction processing system 200, events include TCPA data referring to transaction context 206 and PEA data referring to the PEA 208 for the publishing client. Event Publisher 210 may also add load balancing information into the event. That is, Event Publisher 210 may query the transaction processing system for particular host

machines that have a low load and can include information relating to that low load machine or subscriber in the event in order to direct the event to the low load machine.

Consistent with the distributed object architecture, PEA 208 may look up an Event Publisher 210 that is located on or affiliated with its host machine (as illustrated in FIG. 4), or PEA 208 may look up an Event Publisher 210 anywhere within system 200 using the ORB to locate and communicate with the Event Publisher.

Event Channel 212 implements an event channel that is similar to channels provided in the CORBA Event Service. Any number of Event Channels 212 may be provided and Event Channels may be provided on any machine in the transaction processing system 200 and an Event Publisher 210 may locate and communicate with an Event Channel using ORB services. Event Channel 212 receives an event from an Event Publisher 210 and sends the event on to all Event Subscribers 214 (only one shown). One advantage to providing an event service in parallel with the underlying ORB is that, because the ORB services are minimally used, transaction processing system 200 can easily be configured to run on multiple ORBs. For example, transaction processing system 200 can readily be configured to operate with CORBA or with COM/DCOM or other ORBs known in the art.

Event Subscriber 214 provides a way for classes to "register" for specific events. Event Subscriber 214 receives events from an Event Channel 212 and forwards the events to the appropriate classes. On startup, Event Subscriber 214 contacts an administrator object to obtain a list of all classes and the location of every Event Channel 212 that is defined in the transaction processing system. Event Channel 212 calls each Event Subscriber 214 whenever there is an event in the channel to be processed. Event Subscriber 214 parses the event and determines whether the event is subscribed to by an object on its host. If so, Event Subscriber 214 passes the event to the subscribing object.

If no subscribing object is instantiated on the desired host, Event Subscriber 214 checks to see whether a Subscriber Thread Pool 218 is running for the subscribing class. If not, Event Subscriber 214 starts a new Subscriber Thread Pool 218, typically in a new Java virtual machine in Java implementations, and passes the event to the Subscriber Thread Pool 218. The Subscriber Thread Pool 218 instantiates the subscribing object, such as server object 220, and passes the event to the instantiated subscribing object. The subscribing object then processes the event. In this way, events can be directed to specific machines, either for load balancing purposes or because the event has specific

processing needs that are met by a specific machine, whether or not the subscribing object is running on that machine.

While providing an Event Subscriber 214 on each server host as illustrated in FIG. 4 may be beneficial under some circumstances, consistent with the distributed object architecture described herein, one or more Event Subscribers 214 may be present anywhere within system 200 without regard to the physical structure of the network. That is, Event Subscriber 214 need not be located on server host 242, but only needs to be connected to objects running on server host 242 through an ORB.

For example, when a published event includes load balancing information that specifies a particular server host to process an event, Event Channel 212 sends that event to each Event Subscriber 214 in the system 200 and each Event Subscriber 214 parses the event to determine whether that Event Subscriber is the Event Subscriber that should further process the event. If an Event Subscriber 214 does conclude that it should handle an event, that Event Subscriber passes the event to a subscribing object on the identified host through the ORB. If the subscribing object is not running on that host, Event Subscriber 214 can call Subscriber Thread Pool 218 on that host to instantiate the subscribing object before passing the event on.

A subscribing object, such as server object 220, may also include a Subscriber for Events Agent ("SEA") 216. The SEA class gives the subscribing object an interface to Event Subscriber 214 for receiving events, and also provides a way for the subscribing object to return results to the publishing object. Because the event includes a reference to the PEA 208 of the publishing object, SEA 216 can direct results back from the subscribing object to the publishing object PEA 208 which can then hand the result off to the client.

Server object 220 also joins in the transaction by starting its own TCPA 202 based on the publishing object's TCPA 202 data which includes an IOR for the Transaction Context 206. TCPA 202 allows server object 220 to register as a participant in the transaction, to access data stored in the transaction, to request database services through the Transaction context 206, and to register the server object's vote to commit or roll back the transaction.

The flow of an event through transaction processing system 200 from a single client to a single subscribing object is illustrated by arrows 228, 230, 232, 234, 236, 238. PEA 208 publishes an event at the request of client 204 by first passing 228 the event to

an Event Publisher 210. The Event Publisher 210 then publishes the event to the larger transaction processing system 200 by passing 230 the event, likely by means of an ORB, to an Event Channel 212 that may be running on any machine in the system 200. Event Channel 212 then passes 232 the event on to all Event Subscribers 214. In one embodiment, one Event Subscriber is provided on each host, such as server host 242 for the purpose of directing events to any object located on that host that subscribes to such events. Event Subscriber 214 then passes the event to subscribing objects on its host. If a subscribing object, such as server object 220, is not running on the host, Event Subscriber 214 can call 234 Subscriber Thread Pool 218 to instantiate such an object. The event is then passed 236 to the subscribing object. After the subscribing object performs its logical operations based on the event data, the subscribing object SEA 216 responds to the PEA 208 through a direct call back 238.

The event handling mechanism of transaction processing system 200 has been illustrated using a single subscribing object, however, any number of objects (including zero objects) may subscribe to a given event on any number of machines. Each subscribing object can respond to the publishing object using a distributed call back through its SEA and each subscribing object can participate in the Transaction context 206 through its TCPA. In addition each subscribing object may, as part of the logic it performs as a result of receiving the event, publish its own event or events. Events published by a subscribing object would contain the same TCPA reference data as the original event and would also include a reference to a PEA for the subscribing-turned-publishing object.

An example of event, call back and vote messaging with more than one subscribing object is illustrated in FIG. 5. An exemplary transaction is started by client 204 which creates transaction context 206. Client 204 publishes an event for which S1 252 and S2 254 are subscribing objects. S2 254 registers with transaction context 206, makes any necessary database connections through transaction context 206 and performs whatever logic S2 performs. S2 then calls back 256 client 204 votes 258 with transaction context 206 to commit or roll back the transaction based on whether S2's logical operations were successfully performed and then calls back 256 client 204. S1 also subscribes to the event, registers as a transaction participant with transaction context 206 and acquires any necessary database connections through transaction context 206. In this example, in order for S1 to complete its logical operations, it publishes 260 a new event

to which object S3 262 subscribes. S3 262 then registers as a transaction participant with transaction context 206, makes any necessary database connections through transaction context 206 and performs whatever logic S3 performs. S3 262 then votes 264 with transaction context 206 to commit or roll back the transaction and calls back 266 S1 252 with its results. With results received from S3 262, S1 252 completes its logical operations, votes 268 with transaction context 206 to commit or roll back the transaction and calls back 270 client 204 with its results. Having received results from each of the objects that subscribe to its event, client 204 votes 272 to commit or roll back the transaction and signals transaction context 206 to close the transaction. Transaction context 206 then commits or rolls back the transaction as appropriate based on the voting and reports the result back 274 to the client.

Where Transaction Context 206 tracks voting and database connection information by name as illustrated in FIG. 4A, it is possible for different objects of the same type to share a database connection. For example, if subscribing objects S2 and S3 are objects of the same type, Add_Employee server objects for example, these objects may share the same database connection based on their name entry in Transaction Context 206. This would allow the last Add_Employee server object to act to have the final say as to what data Add_Employee will place in the database. This ensures that the Add_Employee data will not be inconsistent between different Add_Employee server objects in the same transaction. It also allows the later acting Add_Employee server object to have access to uncommitted data placed on the database connection by a prior acting Add_Employee server object, ensuring that the later acting object has access to the most recent data. These objects could also share the same vote in Vote Table 225, however, in many situations, it is preferable for each participant to have its own vote.

As indicated heretofore, aspects of this invention pertain to specific "methods" and "method functions" implementable on computer systems. Those of ordinary skill in the art should readily appreciate that computer code defining these functions can be delivered to a computer in many forms; including, but not limited to: (a) information permanently stored on non-writable storage media (e.g., read only memory devices within a computer or CD-ROM disks readable by a computer I/O attachment); (b) information alterably stored on writable storage media (e.g., floppy disks and hard drives); or (c) information conveyed to a computer through communication media such as telephone

networks. It should be understood, therefore, that such media, when carrying such information, represent alternate embodiments of the present invention.

It will be understood that the foregoing is only illustrative of the principles of the invention, and that various modifications can be made by those skilled in the art without departing from the scope and spirit of the invention.

What is claimed is:

CLAIMS:

1. In a distributed computing environment having a plurality of computers connected by a communications network, a distributed database access system comprising:

a plurality of clients, each client represented by software running on a computer connected to the communications network;

a plurality of databases;

a connection manager maintaining a plurality of database connection pools, each pool maintaining one or more database connections to a database;

wherein, upon request from a client for database access, the connection manager places the client in communication with a database connection selected from a particular database connection pool.

2. The database access system of claim 1, wherein each database connection pool maintains one or more database connections to a single database.

3. The database access system of claim 1, wherein the plurality of databases includes at least one database storing data in a first data storage schema and at least one database storing data in a second data storage schema.

4. The database access system of claim 3, wherein the client request for database access includes a reference to a data storage schema.

5. The database access system of claim 4, wherein the connection manager includes means for converting data storage schema requests into references to database connections to databases that store data in the requested schema.

6. The database access system of claim 5, wherein the first storage schema corresponds to on-line transaction processing storage.

7. The database access system of claim 6, wherein the second storage schema corresponds to on-line analytical processing storage.

8. The database access system of claim 3, wherein the distributed computing environment includes a transaction processing means having a transaction context for registering transaction participants and at least one of the plurality of clients participates in a transaction.

9. The database access system of claim 8, wherein one or more transaction participants request access to a database storing data in the first data storage schema and one or more transaction participants request access to a database storing data in the second data storage schema.

10. The database access system of claim 9, further comprising means for applying a two-phase commitment protocol to commit or roll back a transaction.

11. The database access system of claim 4, wherein when a requested database is unavailable, the connection manager traverses to a backup database of the same type requested.

12. The database access system of claim 2, wherein each database connection pool increases or decreases the number of database connections maintained by the pool in response to client demand for database connections from the pool.

13. In a distributed object computing environment having a plurality of computers connected by a communications network including a plurality of client objects running on the computers, a plurality of databases including at least one database storing data in a first data storage schema and at least one database storing data in a second data storage schema, and a connection manager maintaining a plurality of database connection pools, each pool maintaining one or more database connections to a database, a method for distributed database access comprising the steps of:

a) a client object requests database access, the client request including a reference to said first or second data storage schema;

b) the connection manager passes to the client object a connection to a database storing data in the requested data storage schema.

14. The method of claim 13, wherein the connection manager accesses a registry of object and database connection pool data to obtain a reference to a database connection pool having connections to a database storing data in the requested data storage schema.

15. In a distributed object transaction processing system having:

a plurality of computers connected by a communications network including a plurality of objects running on the computers;

a plurality of databases including at least one database storing data in a first data storage schema and at least one database storing data in a second data storage schema;

a transaction context for registering objects seeking to participate in a transaction; and

a connection manager maintaining a plurality of database connection pools, each pool maintaining one or more database connections to a database;

a method for distributed database access comprising the steps of:

a) a transaction originating object initiating a transaction by initiating a transaction context and registering as a transaction participant with the transaction context;

b) the transaction originating object publishing a message into the distributed object transaction processing system;

c) one or more objects in the distributed object transaction processing system receiving the message and registering with the transaction context as participants;

d) one or more transaction participants placing one or more requests for database access with the connection manager, the requests including a reference to a data storage schema;

e) the connection manager responding to each request by providing a reference to a database connection that fulfills the request to the transaction participant placing the request;

f) each transaction participant that has requested a database connection placing data into the connection;

g) upon closing of the transaction, commitment of the data in the database connections by the transaction processing system.

16. The method of claim 15, wherein at least one database request placed by an object includes a database identifier referring to a database storing data in the first data storage schema and at least one database request placed by an object includes a database identifier referring to a database storing data in the second data storage schema.

17. The method of claim 15, wherein the first data storage schema corresponds to on-line transaction processing storage.

18. The method of claim 17, wherein the second data storage schema corresponds to on-line analytical processing storage.

19. In a system for distributed database access for a distributed object computing environment having a plurality of computers connected by a communications network including a plurality of client objects running on the computers, a plurality of databases including at least one database storing data in a first data storage schema and at least one database storing data in a second data storage schema, and a connection manager maintaining a plurality of database connection pools, each pool maintaining one or more database connections to a database, a computer program product comprising a computer useable medium having computer readable program code to direct the system to perform at least the following steps:

a) a client requests database access, the client request including a reference to said first or second data storage schema;

b) the connection manager passes to the client a connection to a database of the type requested.

20. The computer program product of claim 19, wherein the connection manager accesses a registry of object and database connection pool data to obtain a reference to a database connection pool having connections to a database storing data in the requested data storage schema.

21. In a distributed object transaction processing system having:

a plurality of computers connected by a communications network including a plurality of objects running on the computers;

a plurality of databases including at least one database storing data in a first data storage schema and at least one database storing data in a second data storage schema;

a transaction context for registering objects seeking to participate in a transaction; and

a connection manager maintaining a plurality of database connection pools, each pool maintaining one or more database connections to a database;

a computer program product comprising a computer useable medium having computer readable program code to direct the system to perform at least the following steps:

a) a transaction originating object initiating a transaction by initiating a transaction context and registering as a transaction participant with the transaction context;

b) the transaction originating object publishing a message into the distributed object transaction processing system;

c) one or more objects in the distributed object transaction processing system receiving the message and registering with the transaction context as participants;

d) one or more transaction participants placing one or more requests for database access with the connection manager, the requests including a database identifier;

e) the connection manager responding to each request by providing a reference to a database connection that fulfills the request to the transaction participant placing the request;

f) each transaction participant that has requested a database connection placing data into the connection;

g) upon closing of the transaction, commitment of the data in the database connections by the transaction processing system.

22. The computer program product of claim 21, wherein at least one database request placed by an object includes a database identifier referring to a database storing data in the first data storage schema and at least one database request placed by an

object includes a database identifier referring to a database storing data in the second data storage schema.

23. The computer program product of claim 21, wherein the first data storage schema corresponds to on-line transaction processing storage.

24. The method of claim 23, wherein the second data storage schema corresponds to on-line analytical processing storage.

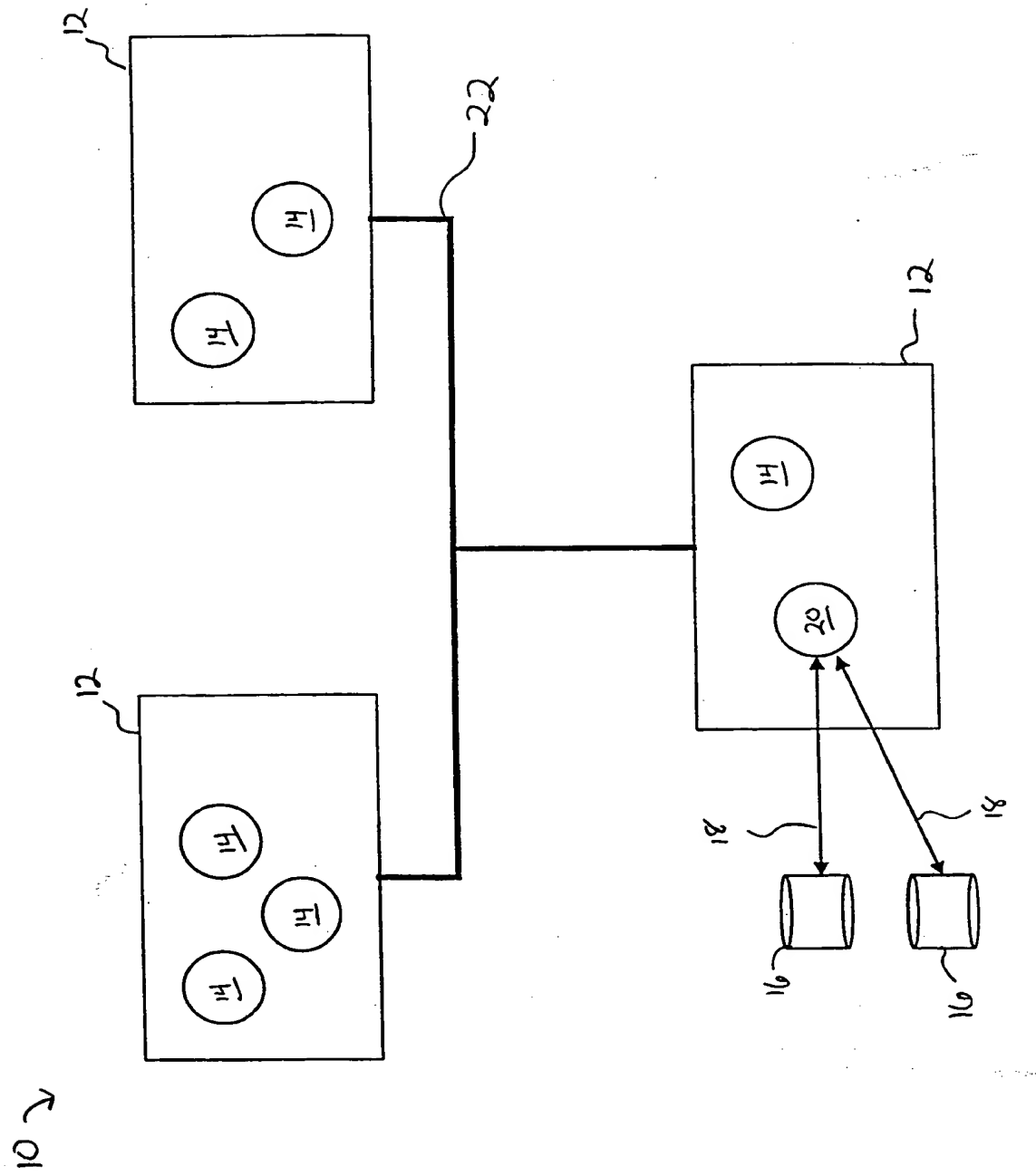


FIG. 1

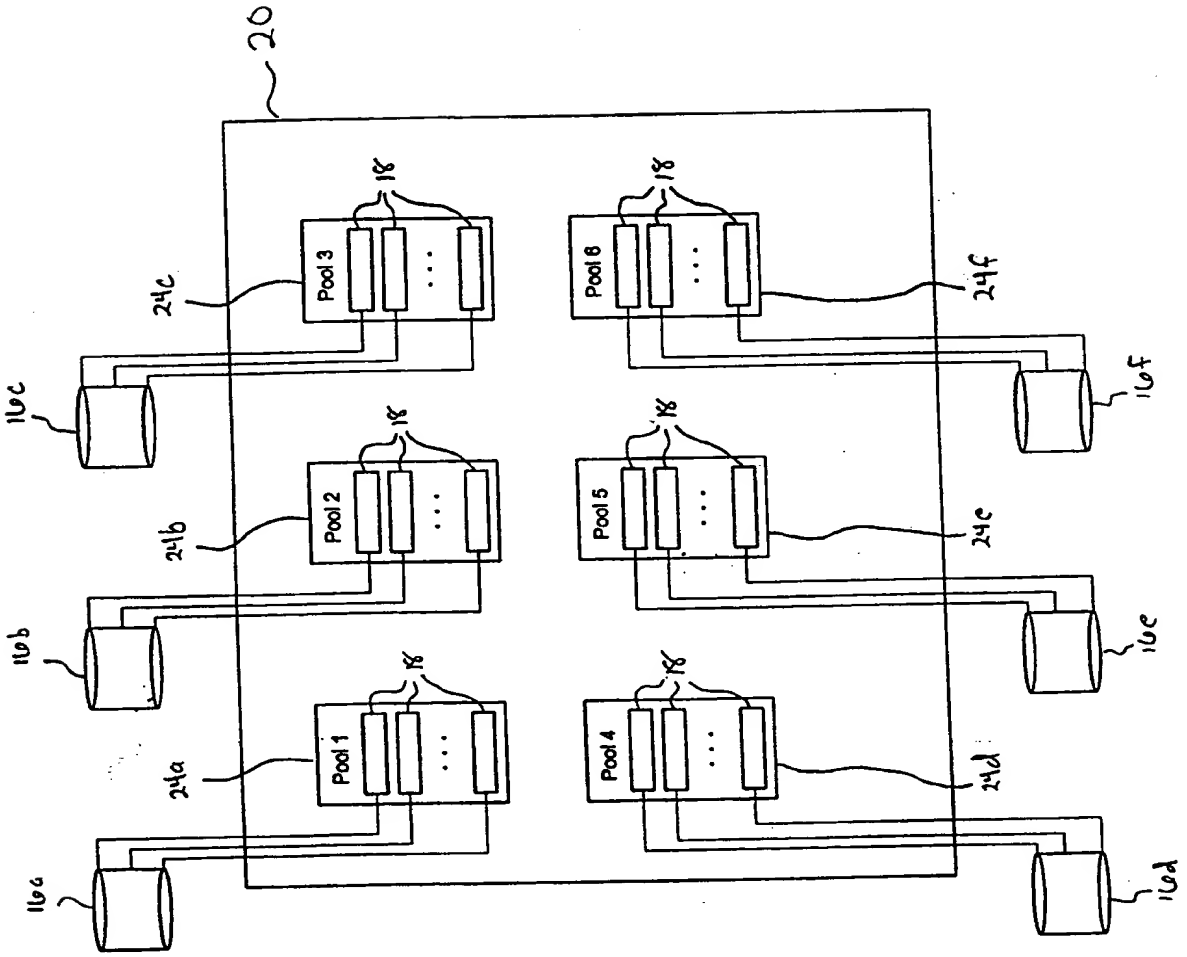

20
18
Pool 3
24c
16c
Pool 6
24f
16f
Pool 2
24b
16b
Pool 5
24e
16e
Pool 1
24a
16a
Pool 4
24d
16d

FIG. 2

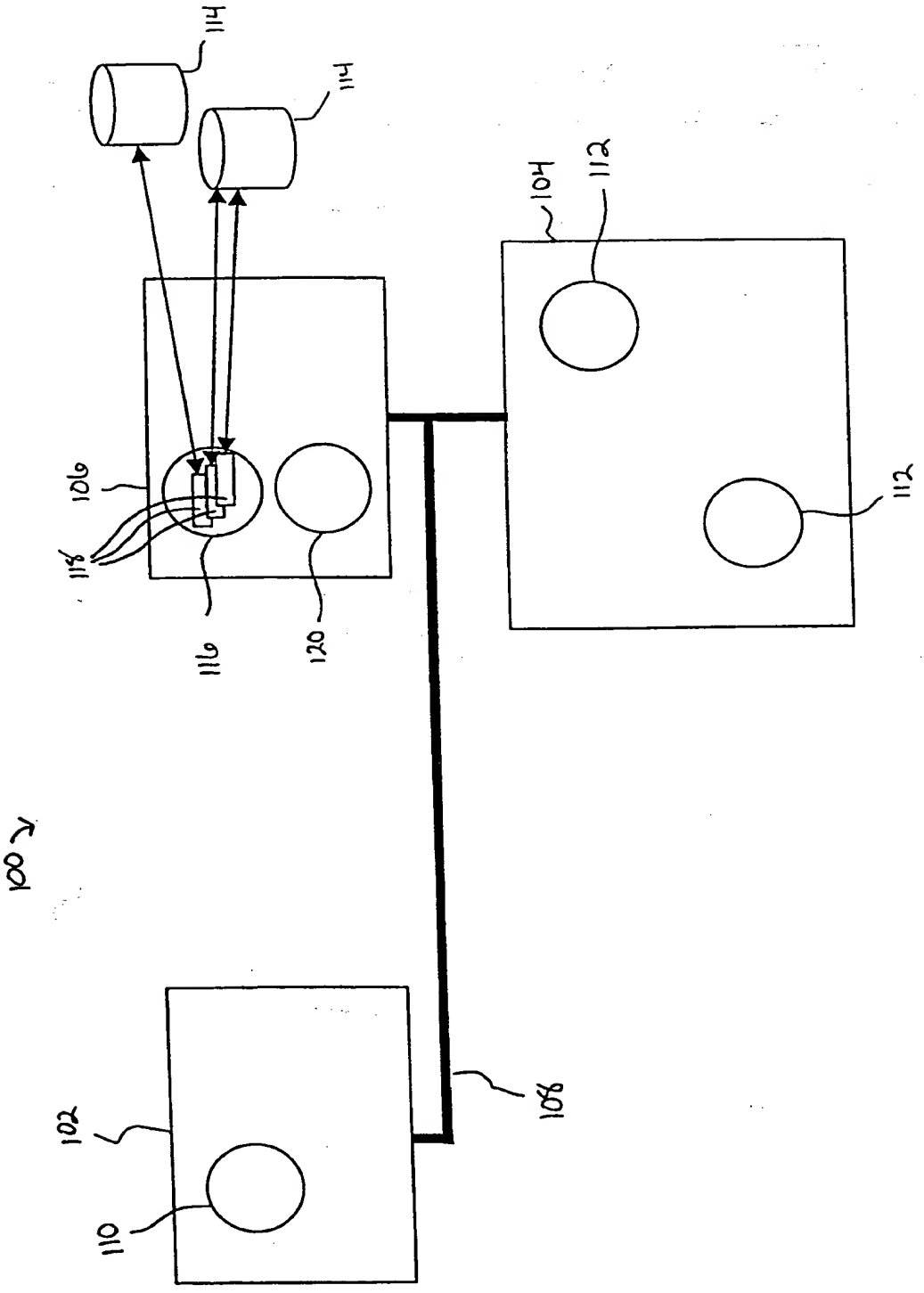


FIG. 3

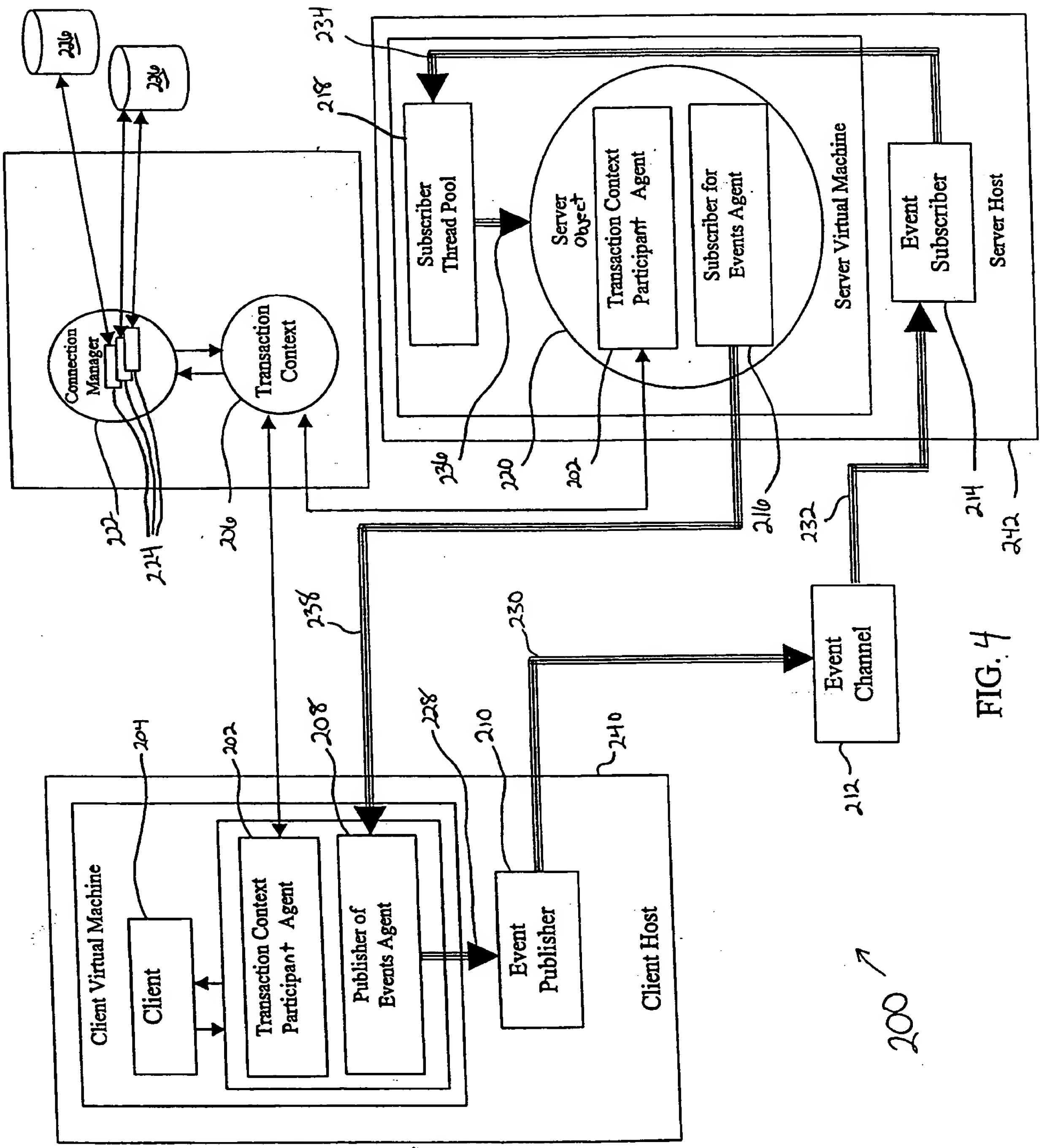

Client Virtual Machine
Client
Transaction Context Participant Agent
Publisher of Events Agent
Event Publisher
Client Host
Event Channel
Connection Manager
Transaction Context
Subscriber Thread Pool
Server Object
Transaction Context Participant Agent
Subscriber for Events Agent
Server Virtual Machine
Event Subscriber
Server Host
200
202
204
208
210
212
214
216
218
220
222
224
226
228
230
232
234
236
238
240
242

FIG. 4

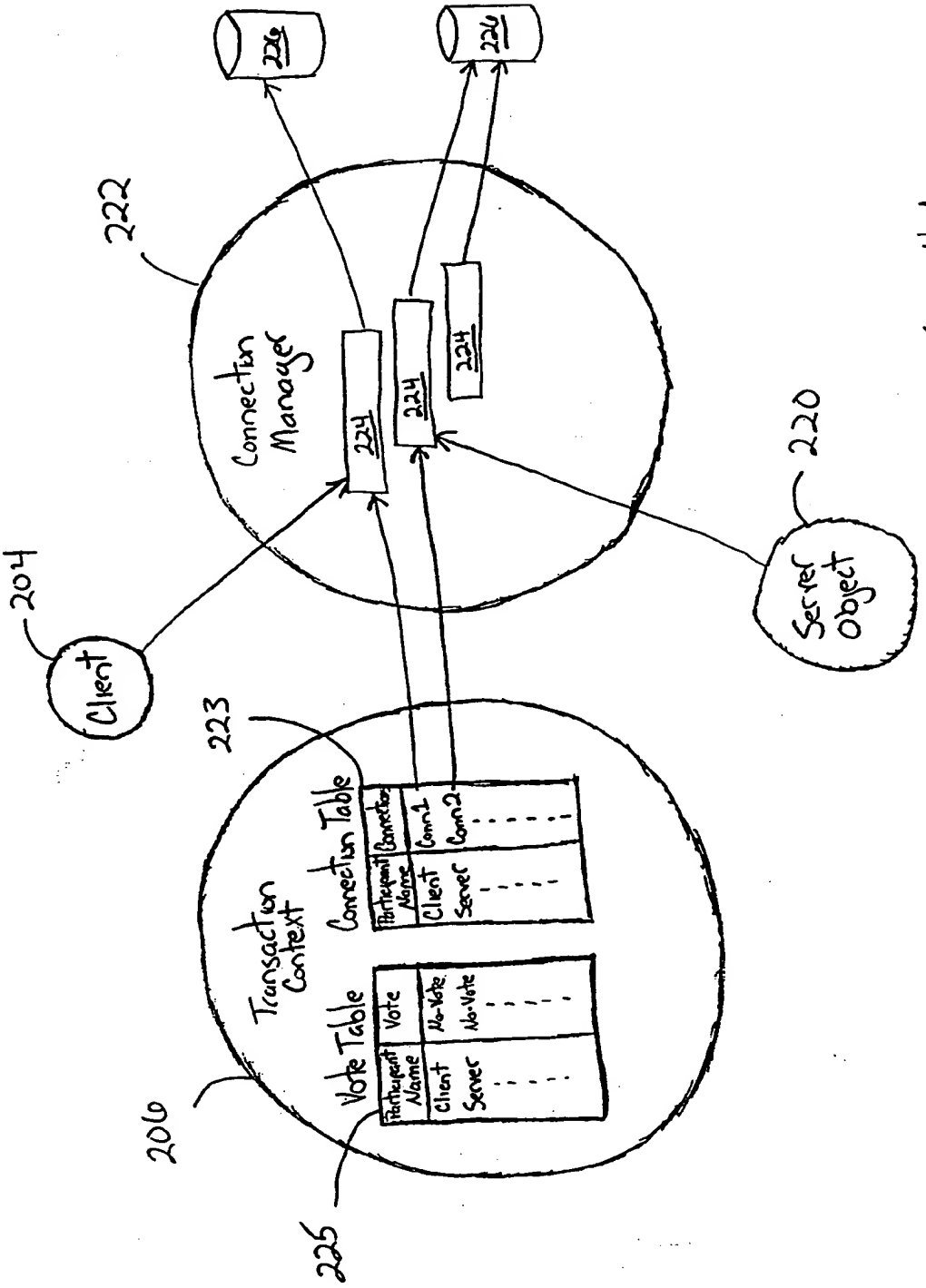

226
226
222
Connection Manager
224
224
224
204
Client
220
Server Object
223
Transaction Context
Connection Table
Participant Name
Connection
Client
Conn1
Server
Conn2
Vote Table
Participant Name
Vote
Client
No-Vote
Server
No-Vote
206
226

FIG. 4A

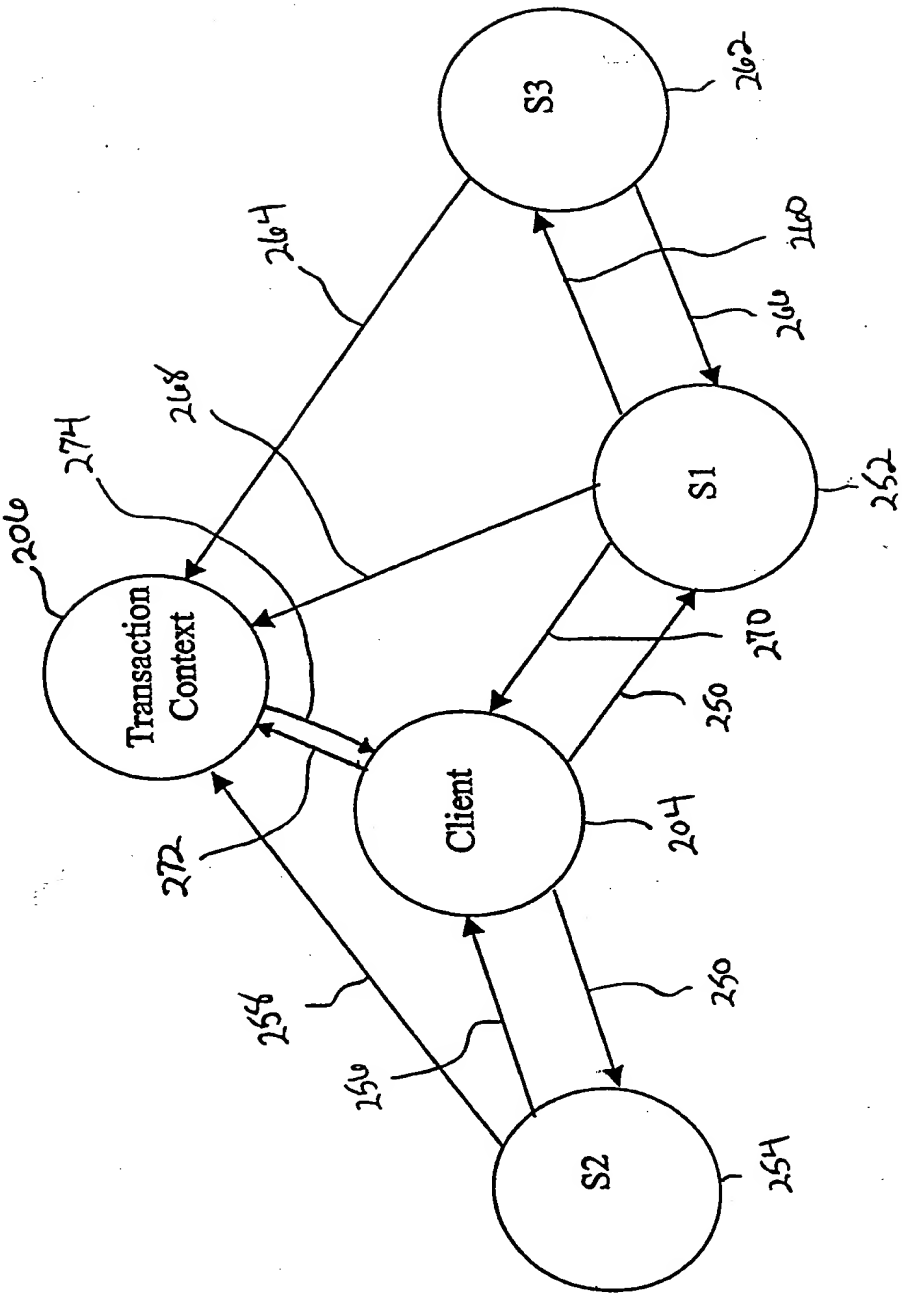

S3
262
264
260
266
268
274
206
Transaction Context
S1
252
270
250
Client
204
272
256
250
258
S2
254

FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/02284

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) ::G06F 15/177, 17/30
US CL :US CL :707/3, 10; 709/203; 717/2
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : US CL :707/3, 10; 709/203

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

WEST, EAST
SCHEMA, DATABASE, STORAGE, CLIENT, COMMUNICATION, NETWORK, REQUEST.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,553,282 A (PARRISH et al.) 03 SEPTEMBER 1996. col. 1, lines 20-67; col. 2, lines 1-65	1-24
A	US 5,659,735 A (PARRISH et al.) A 19 AUGUST 1997, col. 1, lines 22-67; col. 2, lines 1-65	1-14
A	US 5,835,755 A (STELLWAGEN, Jr.) 10 NOVEMBER 1998, col. 1, lines 1-67; col. 2, lines 1-63.	1-24

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 27 APRIL 2000	Date of mailing of the international search report 16 MAY 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer KIM VU Telephone No. (703) 308-6718

Form PCT/ISA/210 (second sheet) (July 1998) ★